SCRATCH NA POWAŻNIE?

Agnieszka Borowiecka, Katarzyna Olędzka
Ośrodek Edukacji Informatycznej i Zastosowań Komputerów
02-026 Warszawa, ul. Raszyńska 8/10
{agnieszka.borowiecka, katarzyna.oledzka}@oeiizk.waw.pl

Abstract. Scratch is an environment very popular among children. However, many shared projects are not advanced. We propose projects which combine math and coding elements. When solving such problems, understanding increases and developing computational thinking is strengthened.

1. Wstęp

Minęły czasy bezwarunkowej fascynacji środowiskiem Scratch, chociaż nadal przyciąga wielu młodych ludzi. Jest językiem programowania, w którym z bloczków – niczym z puzzli – układamy program. Uczeń nie musi uczyć złożonej składni poleceń ani wpisywać ich z klawiatury, działa intuicyjnie. Tworzy więc ciekawe projekty, rozwijając przy tym umiejętności zarówno informatyczne, jak i specyficzne dla danej dziedziny. Poznaje i lepiej rozumie świat, ludzi, a także samego siebie.

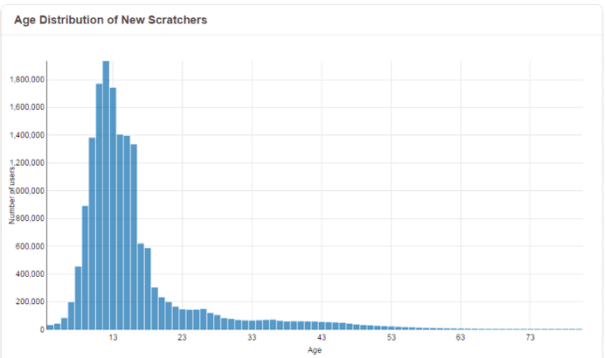
Nie brakuje też głosów krytycznych wśród nauczycieli i użytkowników Scratcha. Prostota języka sprawia, że jest on bardzo ograniczony. Nawet średnio złożony program jest mało przejrzysty, a wielu poleceń po prostu brakuje. Nie na tym jednak skupia się główna fala krytyki. Jednym z założeń metodologicznych, leżących u podstaw tworzenia środowiska Scratch, była nauka programowania dla każdego. Natomiast, jak pokazują badania, większość tworzonych projektów to bardzo proste animacje lub gry, w których elementy programistyczne występują w minimalnym zakresie. Wciąż są podejmowane wysiłki, by pogłębić i poszerzyć wykorzystanie tego środowiska. Chcemy przedstawić kilka pomysłów dotyczących programowania obliczeń za pomocą bloczków.

2. Rozwój myślenia komputacyjnego

Analizując statystyki dostępne na stronie https://scratch.mit.edu zauważamy, że jest już ponad 22 miliony udostępnionych projektów. Powstają one zarówno w wyniku samodzielnej pracy członków społeczności Scratcha, jak i we współpracy.

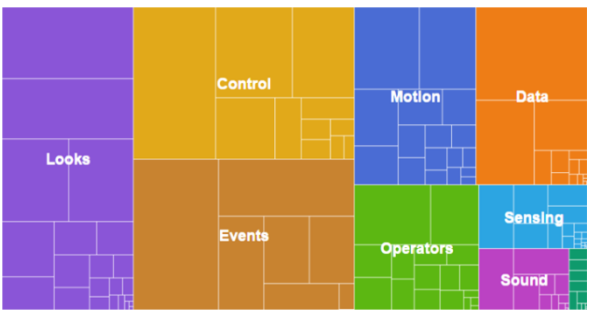
Rysunek 1. Statystyki społecznościowe Scratcha

Według danych widocznych przy rejestracji, przeważają uczniowie w wieku ok. 12 lat. Tworzą i remiksują projekty oraz aktywnie wspierają innych.



Rysunek 2 Wiek użytkowników portalu Scratch

Warto przyjrzeć się analizie rodzajów bloczków zawartych w projektach. W losowej próbie projektów, według danych zawartych na stronie ze statystykami[1], wykorzystane bloczki podzielone według grup: wygląd 22%, kontrola 19%, zdarzenia 19%, ruch 12%, wyrażenia 9%, dane 11%, czujniki 4%, dźwięk 3%, pisak 1%.

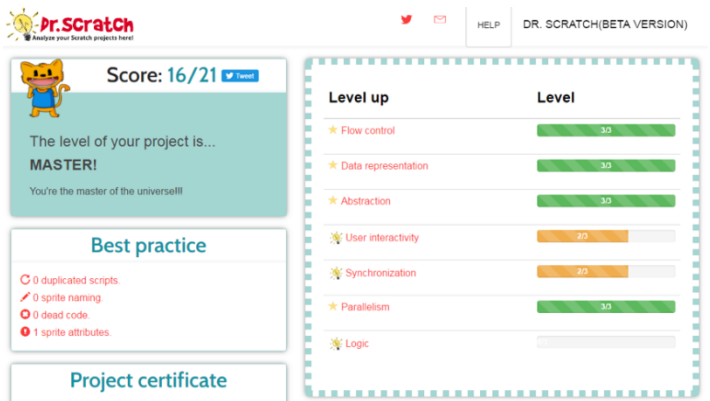


Rysunek 3 Wykorzystanie bloczków w projektach

[1] https://scratch.mit.edu/statistics – ostatni dostęp 22 maja 2017

Rozwój myślenia komputacyjnego wspomaganego programowaniem w Scratchu można rozpatrywać w trzech obszarach: koncepcyjnym, praktycznym i perspektywicznym. Brennan i Resnick [1] do pierwszej z nich zaliczyli stosowanie takich konstrukcji, jak sekwencje poleceń, pętle, współbieżność, zdarzenia, instrukcje warunkowe, operatory i dane. Praktyka związana z myśleniem komputacyjnym dotyczy zarówno samego procesu myślenia, jak i uczenia się, szczególne refleksji o tym, jak się uczymy. W swojej pracy analizują proces powstawania aplikacji w Scratchu na kilku przykładach, zwracając uwagę na metodę projektowania przez kolejne ulepszenia i udoskonalenia, testowanie i poszukiwanie błędów, powtórne wykorzystanie kodu i remiksowanie pracy innych oraz abstrahowanie i modularyzację. W trzecim obszarze zauważyli, że programowanie pomaga w lepszym rozumieniu otaczającego nas świata, ludzi i nas samych. Wzmacnia umiejętność ekspresji, uczenia się z innymi i twórczego stawiania pytań. Praca [1] ma znaczenie teoretyczne – porządkuje zagadnienia związane z rozwojem myślenia komputacyjnego w odniesieniu do nauki programowania w Strachu, jak i praktyczne. W swoich badaniach Brennan i Resnick korzystali z narzędzia wspomagającego automatyczną analizę projektów Scrape User Analysis visualization.

Ciekawym projektem, obecnie rozwijanym, jest Dr. Scratch [3]. Umożliwia on analizę projektu pod względem zastosowanych konstrukcji programistycznych i technik. Między innymi zwraca uwagę na niepotrzebny czy powtarzany kod oraz wykorzystanie i nazewnictwo zmiennych. Raport z analizy kodu aplikacji opatrzony jest komentarzami i wyjaśnieniem dotyczących dobrych praktyk programistycznych.



Rysunek 4 Raport dotyczący wybranego projektu

Należy zdawać sobie sprawę, że automatyczna analiza ma swoje słabe strony, gdyż uczniowie znając kryteria mogą sztucznie zapewniać ich spełnienie. Jednak wartość edukacyjna narzędzia jest nieoceniona – można szybko i precyzyjnie okre-

ślić stopień zaawansowania projektu. Nauczycielowi daje to metodyczne wskazówki do dalszej pracy.

Młodzi ludzie, jak zauważono w artykule [5], wykazują trwałe zainteresowanie nauką programowania. Badania przeprowadzono na grupie osób w wieku 8-18 lat, którzy uczęszczali do Computer Clubhouse. Analizą objęto 536 projektów w Scratchu. Jednym ze spostrzeżeń z badań jest stwierdzenie, że wiele projektów zawiera elementy interakcji z użytkownikiem, a także konstrukcje takie, jak pętle oraz instrukcje warunkowe, a także komunikaty i elementy synchronizacji. Rzadziej można spotkać wykorzystanie zmiennych, elementy logiki oraz losowość. Jak stwierdzają badacze jest to najprawdopodobniej spowodowane tym, że mimo iż są to konstrukcje potrzebne, trudno się ich nauczyć samemu bez wprowadzenia nauczyciela.

Ciekawy eksperyment związany z rozwijaniem myślenia matematycznego w klasach 6 (uczniowie w wieku 11-12 lat) przeprowadził Calao [2]. Porównując wyniki w grupie eksperymentalnej i kontrolnej, łącznie 42 uczniów, doszedł do wniosku, że rozwój myślenia komutacyjnego poprzez programowanie w Scratchu przyczynia się do poprawy wyników uczniów szkół podstawowych w zakresie umiejętności modelowania procesów matematycznych, wnioskowania, rozwiązywania problemów, a jednocześnie wzmacnia motywację. Stwierdził również, że analiza programów i algorytmów jest jedną z mniej rozwiniętych umiejętności w tradycyjnym nauczaniu matematyki, a jednocześnie szczególnie wzmacnianą przez programowanie. Chociaż badania są przeprowadzone na stosunkowo niewielkiej grupie uczniów, stanowią cenną inspirację dla naszych poszukiwań.

W dalszej części przedstawimy własne pomysły dotyczące realizacji zajęć z wykorzystaniem środowiska Scratch, które z jednej strony wspomagają rozwój myślenia komputacyjnego, z drugiej są komputerową realizacją rozwiązań konkretnych problemów. Zajmiemy się algorytmem wypisywania cyfr liczby, rozwiązaniem programistycznym zadania *Gra liczbowa* z konkursu Koala[2] z 2016 r. oraz zagadnieniami związanymi z przeliczaniem systemu dziesiątkowego na dwójkowy i odwrotnie. Pokażemy nie tylko samo rozwiązanie, ale jak do niego stopniowo dojść, zajmując się coraz trudniejszymi problemami.

3. Wypisywanie cyfr liczby

Zadanie polega na wypisaniu kolejnych cyfr liczby, począwszy od ostatniej (najmniej znaczącej) aż do pierwszej. Na przykład dla liczby 2017, wypiszemy 7, 1, 0, 2. Zastanówmy się, jak to zrealizować. Zauważymy, że korzystając z algorytmu dzielenia z resztą, postępowanie dla naszego przykładu można opisać w następujący sposób:

[2] http://vlo.poznan.pl/koala/KOALA2016.pdf

2017 : 10 = 201 reszty 7

201 : 10 = 20 reszty 1

20 : 10 = 2 reszty 0

2 : 10 = 0 reszty 2

Ciąg kolejnych reszt to liczba „przeliterowana" od końca cyfra po cyfrze. Algorytm postępowania możemy zapisać w bardziej formalny sposób:

```
wczytaj wartość zmiennej liczba
powtarzaj aż liczba = 0:
    wypisz resztę z dzielenia liczba przez 10
    przypisz zmiennej liczba wartość liczba podzielona
    całkowicie przez 10
```

Do komputerowej realizacji w Scratchu potrzebujemy zmiennej *liczba* i bloczków *zapytaj* oraz ***odpowiedź*** do komunikacji z użytkownikiem. Zbudowana z bloczków realizacja powyższego algorytmu mogłaby wyglądać następująco:

Rysunek 5 Wypisanie kolejnych cyfr liczby

Zwróćmy uwagę, że w środowisku Scratch nie ma operacji dzielenia całkowitego. Zamiast tego można skorzystać z funkcji **podłoga**, która dla danej liczby obcina jej część ułamkową zostawiając największą liczbę całkowitą mniejszą lub równą podanej wartości.

4. Szczęśliwy numerek

Źródłem ciekawych pomysłów na zajęcia mogą być konkursy matematyczne i informatyczne. Polecamy konkurs Koala i zadanie *Gra liczbowa* z 2016 roku.

Rozważmy następujący problem: w pewnej grze liczbowej drukowane są kupony o sześciocyfrowych numerach. Za „szczęśliwe" uważane są te numery, których suma cyfr stojących na miejscach parzystych jest równa sumie cyfr stojących na miejscach nieparzystych. Na przykład kupon 631752 jest uważany za „szczęśliwy", gdyż

$$6 + 1 + 5 = 3 + 7 + 2 = 12.$$

Zaczniemy od sprawdzenia czy numerek jest szczęśliwy, czy nie. Będziemy postępować podobnie jak w przykładzie poprzednim – rozłożymy liczbę na cyfry, przy czym będziemy zliczać osobno sumy cyfr stojących na pozycjach parzystych i nieparzystych. Wykorzystamy zmienne **parzyste** i **nieparzyste** do liczenia sum cyfr, a zmienną pomocniczą **pozycja** do badania parzystości pozycji kolejnych cyfr liczby. Po zbadaniu wszystkich cyfr liczby wypiszemy informację, czy był to szczęśliwy numer.

Algorytm można zapisać następująco:

```
wczytaj wartość zmiennej liczba
ustaw wartość zmiennej nieparzyste na 0
ustaw wartość zmiennej parzyste na 0
ustaw wartość zmiennej pozycja na 1
powtarzaj aż liczba = 0:
   jeżeli pozycja jest liczbą nieparzystą
      dodaj do nieparzyste resztę z dzielenia liczba
      przez 10
   w przeciwnym przypadku
      dodaj do parzyste resztę z dzielenia liczba przez 10
   wypisz resztę z dzielenia liczba przez 10
   przypisz zmiennej liczba wartość liczba podzielona
   całkowicie przez 10
   zwiększ wartość zmiennej pozycja o 1
jeżeli nieparzyste=parzyste to
   wypisz Szczęśliwy numer!
w przeciwnym przypadku
   wypisz To nie jest szczęśliwy numer!
```

Warto zwrócić uwagę, że do przedniego algorytmu dodaliśmy sumowanie cyfr, osobno dla cyfr znajdujących się na pozycjach parzystych i nieparzystych. Wprowadziliśmy także instrukcję warunkową, która powoduje wypisanie odpowiedniego komunikatu po zakończeniu obliczeń.

Rysunek 6 Badanie czy numer jest szczęśliwy, część 1

Rysunek 7 Badanie czy numer jest szczęśliwy, część 2

Przedstawione rozwiązanie można użyć do sprawdzenia, czy dowolna liczb jest szczęśliwym numerem kuponu.

5. Szczęśliwe kupony

Postawmy pytanie takie, jak w oryginalnej treści zadania: Ile jest „szczęśliwych” kuponów o numerach od 000000 do 999999?

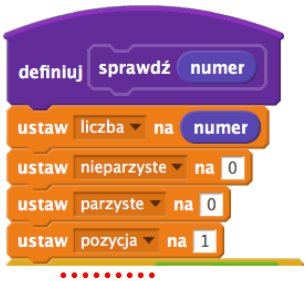
Potrafimy już badać czy dany numer jest szczęśliwy, czy też nie. Pozostaje nam zliczyć wszystkie „szczęśliwe" numery. Możemy to zrobić w następujący sposób:

```
ustaw wartość zmiennej ile_szczęśliwych na 0
ustaw wartość zmiennej kandydat na 0
powtórz 1000000 razy
   sprawdź czy numer kandydat jest numerem szczęśliwym,
   jeśli tak, to zwiększ wartość zmiennej ile_szcześliwych o 1
   zwiększ wartość zmiennej kandydat o 1
wypisz ile_szczęśliwych
```

Badamy wszystkie liczby od 000000 do 999999, jest ich 1000000. Liczbę numerów szczęśliwych pamiętamy na zmiennej **ile_szczęśliwych**. W środowisku Scratch nie ma możliwości definiowania funkcji, której wynikiem jest liczba. Utworzymy nowy blok o nazwie **sprawdź**, który będzie sprawdzał podany numer i jeśli jest on szczęśliwy, zwiększy wartość zmiennej **ile_szczęśliwych** o 1.

Rysunek 8 Zliczanie szczęśliwych kuponów

Poniżej treść zdefiniowanego przez nas bloku **sprawdź**, który dla danego numeru sprawdza czy jest on „szczęśliwy".



Rysunek 9 Badanie, czy numer jest szczęśliwy, część 1

Rysunek 10 Badanie, czy numer jest szczęśliwy, część 2

Efektem działania aplikacji jest policzenie „szczęśliwych" kuponów o numerach od 000000 do 999999. Okazuje się, że jest ich dokładnie 55 252. Taki sam wynik można otrzymać nie pisząc programu, jak to zostało zaproponowane, ale rozważając problem teoretycznie jako zadanie kombinatoryczne.

Matematyczne zmagania, poza komputerową realizacją, są bardzo ciekawe i zachęcamy czytelników do własnych prób. W podpowiedzi warto zauważyć, że zadanie można rozbić na podproblemy:

1) liczby są zapisane za pomocą sześciu cyfr – trzy cyfry odpowiadają pozycjom parzystym, trzy nieparzystym;
2) suma trzech cyfr jest liczbą z przedziału 0 – 27;
3) liczba kombinacji do uzyskania 0 i 27 jest ta sama, podobnie dla 1 i 26, itd. Dokładniej, każda trzycyfrowa liczba dająca określoną sumę składa się z trzech różnych cyfr (6 kombinacji), 2 różnych cyfr (3 kombinacji) lub wszystkich jednakowych cyfr (1 kombinacja). Na przykład
   - dla 3: są to sumy postaci: 1+1+1 (1 kombinacja), 2+1 (6 kombinacji), 3 (3 kombinacje);
   - dla 4: 1+1+2 (3 kombinacje), 2+2 (3 kombinacje), 3+1 (6 kombinacji), 4 (3 kombinacje).

Wszystkie możliwe kombinacje przedstawia tabelka poniżej.

suma	0	1	2	3	4	5	6	7	8	9	10	11	12	13
liczba kombinacji	1	3	6	10	15	21	28	35	45	55	63	69	73	75
suma	27	26	25	24	23	22	21	20	19	18	17	16	15	14

4) Jeżeli daną sumę trzech cyfr można przedstawić na x sposobów, to w naszym zadaniu możliwych kombinacji jest x2, x – dla pozycji parzystych i x dla pozycji nieparzystych. Wobec czego po zsumowaniu wszystkich kwadratów liczb z tabeli otrzymujemy 55 252.

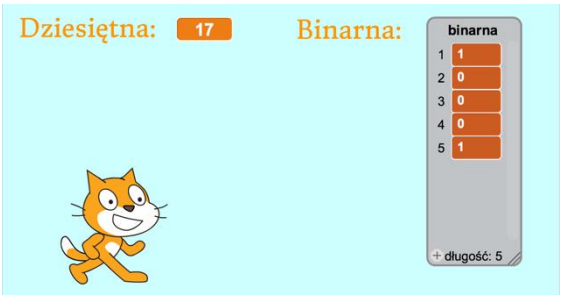
6. Dec to bin i odwrotnie

Rozważmy najpierw zadanie przeliczania liczb zapisanych w systemie dziesiętnym na liczby zapisane w systemie binarnym. Algorytm ten jest bardzo podobny do algorytmu wypisywania liczby cyfra po cyfrze. Wczytujemy liczbę, a następnie w pętli wypisujemy resztę z dzielenia przez 2.

wypisywanie cyfra po cyfrze	**przeliczanie na system binarny**

wczytaj wartość zmiennej **liczba**
powtarzaj aż **liczba = 0***:*
  wypisz resztę z dzielenia **liczba** *przez 10*
  przypisz zmiennej **liczba** *wartość* **liczba** *podzielona całkowicie przez 10*
	wczytaj wartość zmiennej **liczba**
	powtarzaj aż **liczba = 0***:*
	  wypisz resztę z dzielenia **liczba** *przez 2*
	  przypisz zmiennej **liczba** *wartość* **liczba** *podzielona całkowicie przez 2*

Tym razem duszek zamiast wypowiadać kolejne cyfry liczby będzie dopisywać je do listy, a gotową listę wyświetlimy na ekranie.



Rysunek 11 Liczba przeliczona na system binarny

Liczbę podaną przez użytkownika zapamiętujemy w zmiennej **dziesiętna**. Utworzymy także listę **binarna**, która będzie zawierała kolejne cyfry zapisu podanej liczby w systemie dwójkowym. Zamiast pokazywać cyfry rozwinięcia binarnego liczby poleceniem **powiedz**, dodajemy je do listy **binarna** za pomocą bloczka **dodaj do listy**:

Należy pamiętać o usunięciu zawartości listy przed rozpoczęciem zamiany nowej liczby na system dwójkowy. Skrypt przeliczający podaną wartość na system dwójkowy będzie wyglądał następująco:

Rysunek 12 Przeliczanie wartości na system binarny

Warto zwrócić uwagę na kolejność cyfr na liście – pierwsza cyfra na liście jest ostatnią cyfrą otrzymanej liczby, druga – przedostatnią, itd. Możemy także otrzymać listę w odwrotnej kolejności – wystarczy zmienić jeden bloczek w skrypcie dodający kolejne wartości do listy:

Pracując z gotowym programem możemy przeprowadzić z uczniami dyskusję na temat podobieństw i różnic w zapisie liczb w obu systemach, zadając na przykład następujące pytania:

- Ile cyfr potrzebnych jest do zapisania danej liczby w systemie binarnym?
- W którym systemie zapis liczby jest dłuższy i dlaczego?
- Jaką największą liczbę można zapisać za pomocą 8 cyfr binarnych?

Pracując ze starszymi i bardziej zaawansowanymi programistycznie uczniami możemy dodać do projektu przedstawienie liczby binarnej za pomocą duszków ilustrujących poszczególne cyfry.

Proponujemy zacząć od prostszej wersji projektu. Najpierw tworzymy cztery duszki pozwalające przedstawiać zapis binarny liczb z zakresu od 0 do 15. Każdy duszek ma dwa kostiumy, pierwszy to cyfra 0, drugi – cyfra 1. Początkowo ustawiamy kostiumy wszystkich duszków na 0. Następnie przeliczamy podaną przez użytkownika liczbę na system dwójkowy. Jeżeli podczas dzielenia przez 2 otrzymamy jedynkę, to zmieniamy kostium odpowiedniego duszka-cyfry.

Rysunek 13 Ilustaracja graficzna kiczby binarnej

Do przedstawienia liczb dowolnej długości możemy wykorzystać klonowanie duszków. Definiujemy duszka-prototyp przypisując mu dwa kostiumy, zero i jeden. Dodajemy zmienną prywatną **numer**, służącą do ustalenia pozycji klona w liczbie binarnej. Pierwszy klon będzie odpowiadał cyfrze najmniej znaczącej i będzie miał numer 1, kolejny odpowiada 2^1 z numerem 2, następny 2^2 z numerem 3 itd.

Po wypełnieniu listy **binarna** kolejnymi cyframi zapisu binarnego liczby, generujemy tyle klonów, ile jest pozycji na liście. Każdemu klonowi ustawiamy właściwy kostium i ustalamy położenie na ekranie.

Rysunek 14 Tworzenie klonów duszka-cyfry

Projekt przeliczający liczby z systemu dwójkowego na dziesiętny także możemy przygotować w dwóch wersjach. Prostszy projekt pozwala badać liczby składające się maksymalnie z 4 cyfr (mniejsze od 16). W projekcie bardziej złożonym liczba cyfr binarnych może być dowolna, ograniczona jedynie czytelnością zapisu i miejscem w oknie Scratcha. W obu przypadkach wygodnie wykorzystać zmienne do przechowywania informacji o poszczególnych cyfrach zapisu binarnego lub odpowiadających im liczbach dziesiętnych. Na przykład:

$1111_{(2)} = 1*2^3 + 1*2^2 + 1*2^1 + 1*2^0 = 8 + 4 + 2 + 1 = 15_{(10)}$

zmienna	**dana1**	**dana2**	**dana3**	**dana4**
wartość binarna	1	1	1	1
wartość dziesiętna	1	2	4	8

Podstawowy skrypt, jaki należy dodać do projektu, będzie zmieniał kostium po kliknięciu w duszka – odpowiednio na 0 lub 1. Następnie policzymy nową wartość liczby w systemie dziesiętnym. W obliczeniach wykorzystujemy fakt, że w Scratchu można odwoływać się do kostiumów duszka za pomocą ich numeru, przy czym numeracja rozpoczyna się od 1.

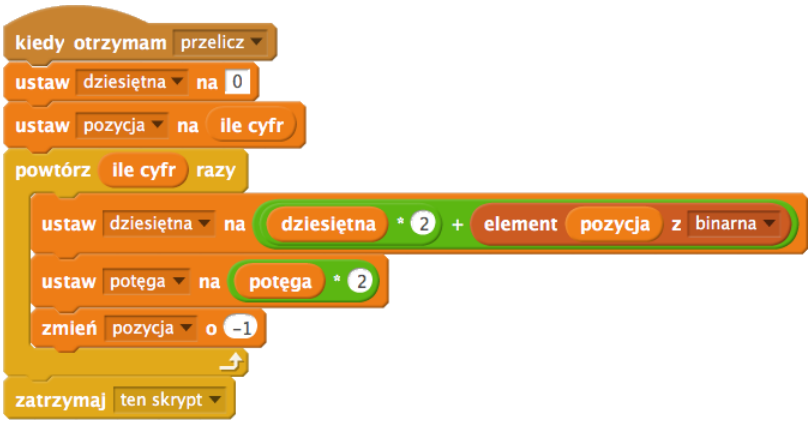
Kliknięty duszek może od razu policzyć nową wartość zmiennej **dziesiętna** lub wysłać komunikat do innych duszków o konieczności dokonania przeliczenia. Poniżej przykładowe skrypty reagujące na kliknięcie w duszka i liczące wartość zmiennej **dziesiętna**:

Rysunek 15 Przeliczanie liczby na system dziesiętny – kliknięcie w duszka

7. Podsumowanie

Młodzi ludzie wiele czasu spędzają w cyfrowym świecie. Są odbiorcami, ale i twórcami. Zadaniem nauczycieli jest towarzyszyć uczniom w ich zmaganiach.

Nawet jeśli pierwsze kroki w Scratchu to próby stworzenia własnej gry, można im postawić dalsze, bardziej wymagające zadania. Ważną rolą nauczyciela jest odpowiednio dopasować ćwiczenia do możliwości uczniów, ale także do ich potrzeb. Na początku wykorzystajmy wizualne środowisko, by ich wprowadzić w świat programowania i pokazać podstawowe techniki algorytmiczne.



Rysunek 16 Przeliczanie liczby na system dziesiętny

Programując z uczniami działania na liczbach naturalnych czy analizując sposoby reprezentowania w komputerze wartości, nie tylko realizujemy zapisy podstawy programowej. Pokazujemy jak na konkretnych przykładach rozwiązywać problemy z różnych dziedzin, szczególnie z matematyki, ze świadomym wykorzystaniem metod i narzędzi wywodzących się z informatyki. Jednocześnie pozwala to na lepsze zrozumienie, jakie są obecne możliwości technologii, komputerów i ich zastosowań. Według zapisów w nowej podstawie programowanej jako nauczyciele informatyki, powinniśmy położyć nacisk na rozwój myślenia komputacyjnego. Dostarcza one metod rozwiązywania problemów, które mogą być wykorzystane w poznawaniu świata, ludzi i siebie. Każdy, kto ma za sobą pierwsze kroki w programowaniu, wie jak potrzebna jest cierpliwość i wytrwałość.

Literatura

1. Brennan K., Resnick M., *New frameworks for studying and assessing the development of computational thinking*, http://web.media.mit.edu/~kbrennan/files/Brennan_Resnick_AERA2012_CT.pdf, ostatni dostęp 22.05.2017 roku.
2. Calao L. A., Moreno-León J., Correa H. E., Robles G., *Developing Mathematical Thinking with Scratch An Experiment with 6th Grade Students*, https://www.researchgate.net/publication/282861505_Developing_Mathematic

al_Thinking_with_Scratch_An_Experiment_with_6th_Grade_Students, ostatni dostęp 22.05.2017 roku.

3. Dr. Scratch: Automatic Analysis of Scratch Projects to Assess and Foster Computational Thinking, http://www.drscratch.org, ostatni dostęp 22.05.2017 roku.
4. Jochemczyk W., Olędzka K., *Ważenie a system binarny*, Informatyka w Edukacji, Toruń 2016.
5. Maloney J., Peppler K., Kafai Y.B., Resnick M., Rusk N., *Programming by Choice: Urban Youth Learning Programming with Scratch*, https://www.cs.swarthmore.edu/~turnbull/cs91/f09/paper/maloney08.pdf, ostatni dostęp 22.05.2017 roku.
6. Perekietka P., *Informatyka unplugged (bez komputera) w kształceniu myślenia komputacyjnego*, Informatyka w Edukacji, Toruń 2016.
7. Sysło M.M., *Myślenie komputacyjne. Nowe spojrzenie na umiejętności informatyczne,* Informatyka w Edukacji, Toruń 2014.
8. Sysło M.M., *Wprowadzając... porządek*, Informatyka w Edukacji, Toruń 2016.